Tooling Support for Variability and Architectural Patterns in Systems Engineering

Thomas Degueule, Joao Bosco Ferreira Filho, Olivier Barais, Mathieu Acher, Jérôme Le Noir, Sébastien Madelénat, Grégory Gailliard, Godefroy Burlot, Olivier Constant

Tooling Support for Variability and Architectural Patterns in Systems Engineering

Thomas Degueule, Joao Bosco Ferreira Filho, Olivier Barais and Mathieu Acher
Inria - IRISA
Université de Rennes 1

Jérôme Le Noir and Sébastien Madelénat
Thales Research & Technology
Palaiseau, France

Grégory Gailliard and Godefroy Burlot
Thales Communications & Security
Genneviliers, France

Olivier Constant
Thales Global Services
Vélizy-Villacoublay, France

ABSTRACT

In systems engineering, the deployment of software components is error-prone since numerous safety and security rules have to be preserved. Furthermore, many deployments on different heterogeneous platforms are possible. In this paper we present a technological solution to assist industrial practitioners in producing a safe and secure solution out of numerous architectural variants. First, we introduce a pattern technology that provides correct-by-construction deployment models through the reuse of modeling artifacts organized in a catalog. Second, we develop a variability solution, connected to the pattern technology and based on an extension of the common variability language, for supporting the synthesis of model-based architectural variants. This paper describes a live demonstration of an industrial effort seeking to bridge the gap between variability modeling and model-based systems engineering practices. We illustrate the tooling support with an industrial case study (a secure radio platform).

1. INTRODUCTION

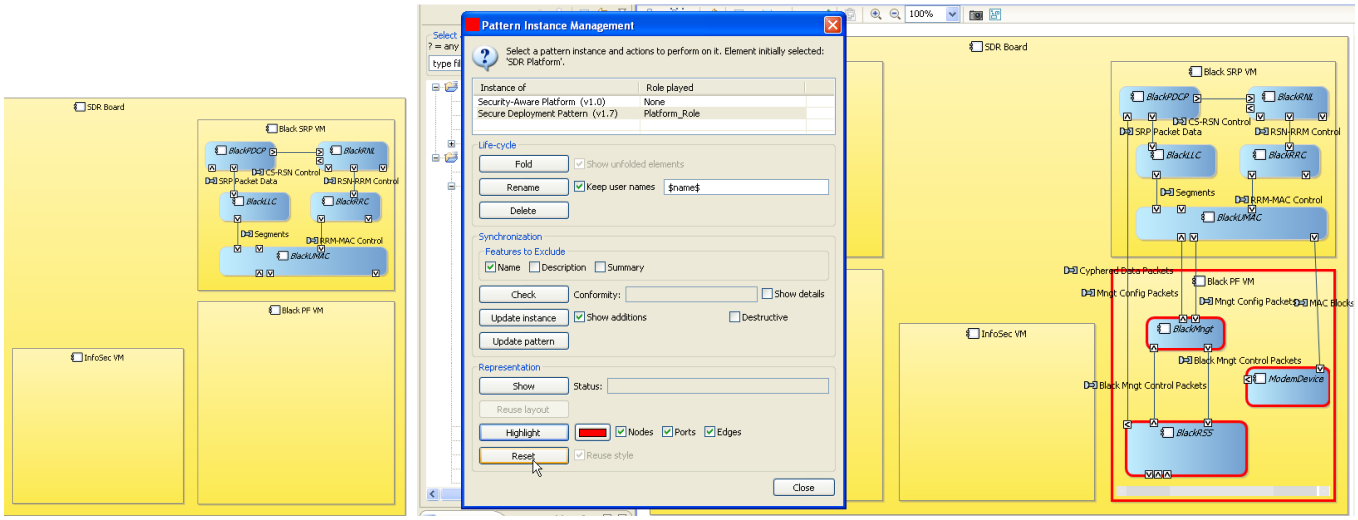
The design of complex systems challenges engineers to deal with massive pieces of software – a typical contemporary car has about 100 million lines of code. A divide and conquer approach is usually employed, involving a multiplicity of stakeholders and expertises: each concern of the system is engineered separately through the use of several domain-specific languages and specialized tooling support. For instance, the ISO/IEC 42010 – a standard addressing descriptions of software-intensive system architectures – recommends the adoption of such practice [5]. The standard introduces a general-purpose framework, in which various techniques and architecture description languages can be applied. In the context of Thales, practitioners rely on the Capella open source project [14] that provides an implementation of a multi-viewpoints systems engineering workbench.

The architecture description is organized into multiple *architectural models*. Each model represents the target system from a particular perspective while addressing one or more stakeholders' concerns. Eventually the final architecture must consider all functional and non-functional requirements, including *design rules*.

In the context of systems engineering, the deployment of software components is usually error-prone since numerous safety and security rules have to be preserved at the architecture level. A manual elaboration of an architectural model most likely leads to the violation of design rules. Furthermore, industrial practitioners should usually reiterate the process when the assessment of some possible architectures (variants) does not produce the expected results. Another related issue that exacerbates the problem is that numerous deployments on different platforms are actually possible. In general, organizations need to construct slightly different variants of a same system for addressing new requirements. The support of variability is also important for exploring and assessing different architectural alternatives.

We observed this practical difficulty in the context of Thales and the use of Capella [14]. Overall, the exploration and justification of an architectural solution (variant) was ad-hoc, manual, and mainly consisted in a series of tries and errors on the modeling assets. For addressing previous limitations, it is necessary to (1) *reuse* as much as possible modeling assets and know-how – stakeholders cannot restart from scratch each time a system is plan to be deployed in a given context; (2) *automate* the derivation of new architectures for avoiding accidental complexity; (3) model and support *variability* for synthesizing and exploring architectural variants.

This paper introduces a technological solution to assist industrial practitioners in producing a safe and secure solution out of numerous architectural variants. We introduce mechanisms to combine variability modeling and the notion of model component (called *"pattern"* hereafter) in an industrial context. We first introduce a pattern technology that provides correct-by-construction deployment models through the reuse of modeling artifacts organized in a catalog. We then present a variability solution, connected



(a) Original SDR architecture

(b) Application of a secure deployment pattern

to the pattern technology and based on an extension of the Common Variability Language (CVL), for supporting the derivation of model-based architectural variants. This approach brings two benefits: i) it provides a clear operational semantics for fragment substitution based on the operational semantics of pattern application in models; ii) it provides a variability modeling integration with systems engineers habits based on the reuse of model fragments and patterns.

The targeted audience is researchers or industrials working in systems engineering, safety and security design, model-based development, and product lines. It can also be of interest for tool builders as we integrate variability mechanisms to an industrial environment.

This paper describes a live demonstration of an industrial effort seeking to tool practitioners and bridge the gap between variability modeling and systems engineering practices. Section 2 introduces the pattern technology and its support for reusing model fragments and rules. Section 3 presents our extension of CVL as an algebra to orchestrate the patterns integration. We illustrate the technologies through a demonstration of a real system engineering case study (a secure radio platform) where variability resides both at the hardware and software levels and must be correctly combined. Section 4 discusses some related work and Section 5 concludes the paper.

2. PATTERNS FOR REUSE

Designing large systems often involves repetitive modeling tasks. Certain modeling principles and know-how, usually based on domain expertise, have to be applied in different parts of the model under construction, or reused throughout different models. These modeling principles or *patterns* can simply be a predefined set of model elements, specific modeling rules, or a combination of both. Making these patterns explicit foster their reuse in different models. By enabling practitioners to leverage previous engineering efforts, patterns increase productivity and help to enforce modeling rules in order to guarantee that different models keep conforming to certain business-specific criteria.

The Pattern component of the EMF Diff/Merge[1] project provides such a support for defining and reusing patterns. In a nutshell, a pattern can be created from model elements and stored in a *catalog*. These original model elements become the first *instance* of the pattern. The pattern can then be applied somewhere in the same model or in a different one. This gives birth to a new instance of the pattern. Patterns and instances have their own separate life cycles, but they can be synchronized whenever needed. Concretely, it is possible to check that an instance still conforms to its pattern and have an overview of the differences. If there are differences, the instance can be updated according to the last version of the pattern. Conversely, every instance can also be used for updating the original pattern.

Roles are the explicit integration points of a pattern. Integration refers to how pattern elements are inserted in a model when the pattern is being applied. Roles support two insertion modes:

Addition pattern elements are stored in some container element in the model;

Merge pattern elements are merged with existing model elements in the model. The sub-elements and properties of the model element and the pattern element are combined, giving precedence to the pattern elements.

When a pattern is being applied, each of its roles is associated to one or several elements of the user model. These elements are used either for merging the pattern elements mapped to the role or as a container for storing them.

Figure 1a shows an excerpt of the architecture model for a software-defined radio (SDR) that will be used throughout the demonstration. The SDR is composed of several components with different safety and security requirements that must be physically isolated on different boards or logically isolated on different virtual machines (VM). Patterns are used to capture the domain-specific knowledge of engineers in safe and secure deployment strategies. Typically, different deployment strategies are captured in different patterns, facilitating their application on different projects. Figure 1b depicts the application of a secure deployment pattern and the resulting deployment model. The pattern technology allows to explore different possible deployment strategies depending on the roles played by the different components and the configuration of the pattern application (*e.g.* number of boards and VMs, safety level, etc.).

[1] http://wiki.eclipse.org/EMF_DiffMerge/Patterns

3. VARIABILITY + PATTERNS

The patterns allow practitioners to reuse model fragments for a variety of situations. We equip the pattern technology with variability support so that numerous architectural variants can be automatically derived out of a base model. The key idea is to pilot the application of corresponding patterns through a feature model.

We rely on the *Common Variability Language (CVL)*. CVL is a domain-independent language for specifying and resolving variability over any instance of any MOF compliant [11] technology. The overall principle of CVL is close to many product line approaches: (i) A feature model formally represents features/decisions and their constraints, and provides a high-level description of the product line (domain space); (ii) A variability realization model (VRM) containing the mapping relationships between the feature model and the domain artifacts; (iii) The base models (BM) serving as core assets to be varied and form new derived models conforming to a domain-specific modeling language.

We develop KCVL[2] an implementation of CVL augmented with specific concepts needed for realizing patterns. KCVL is bundled as a set of Eclipse plugins and consists of a set of integrated components that support the different aspects of our approach:

- A textual editor, implemented with Xtext [6], that allows to express in a concise syntax the different parts of a CVL model: the feature model, the variability realization model, the modeling artifacts, and the configuration model. It embeds handy facilities such as autocompletion on base model elements and basic static semantics checking (see Figure 2);
- A derivation engine that accepts as input a valid CVL model and derive appropriate variants of the base models depending on the configuration choices expressed in the configuration model;
- A binding to FAMILIAR [**?**, 1] for manipulating and reasoning about (multiple) feature models;
- New types of CVL variation points dedicated to the manipulation of patterns, and their instantiation on base models. Specifically, we develop **PatternIntegration** for the addition semantics and **PatternFusion** for the merging semantics, thus completing basic variation points of CVL for activating/deactivating model elements. They can be seen as high-level constructs for weaving model components, with specific semantics based on Thales engineers' expertise. They are realized by binding (see **RoleBinding**) a set of base model elements to the different roles of a pattern. When selected, these variation points apply the corresponding pattern on concrete elements of the base model.

As a concrete example, Figure 2 depicts the textual editor for a KCVL file describing a subset of the VRM for the SDR system. The different roles of the secure deployment pattern are binded to concrete elements of the base model and the **PatternFusion** variation point is associated to a specific choice in the associated feature model. As shown in Figure 3, a specific configuration of the VAM can then be selected, and the target model automatically derived.

[2] http://diverse-project.github.io/kcvl/

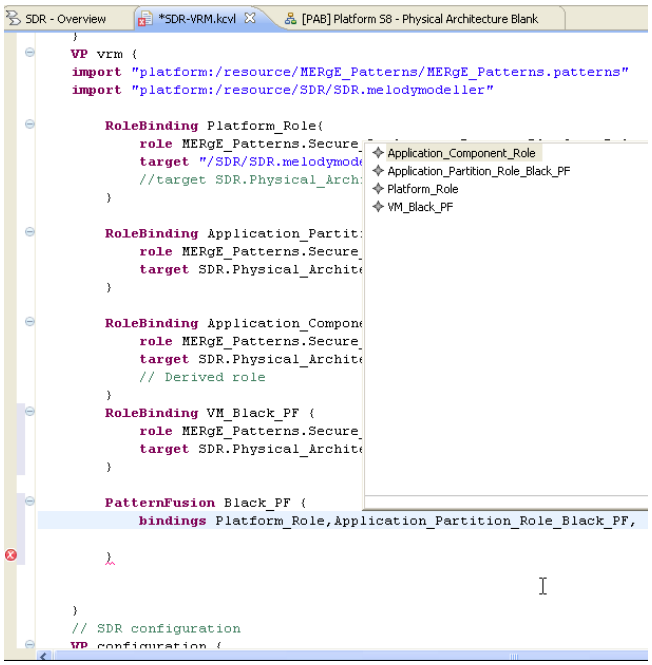


Figure 2: KCVL in action: specifying a VRM

```
// SDR configuration
VP configuration {
    resolution for VMPartition = true
        resolution for Red_PF_VM = false
        resolution for Black_PF_VM= true
        resolution for InfoSec_VM = false
        resolution for Red_SRP_VM = false
        resolution for Black_SRP_VM =false
}
```

Figure 3: KCVL in action: specifying configuration choices

KCVL has been designed in order to manipulate any MOF-compliant base model. As a result, it has been successfully experimented to configure base models expressed in different formalisms such as Ecore, UML or Capella [14]. Additionally, KCVL takes into account the specific semantics of the languages that are used to express the base models. For example, when an element is removed by a variation point, all the elements or references of the base models that are impacted by this change are updated accordingly [7].

KCVL seamlessly integrates with several tools developed within the Eclipse Modeling Project initiative, including EMF diff/merge. Therefore, the new patterns that may be inserted in the base models benefit from all the facilities presented in Section 2: catalogs and instances management, conformance checking and synchronization. By combining different tools in an integrated workflow, KCVL serves as a one-stop shop for variability management in model-based systems architecture.

4. RELATED WORK

This paper pursues our previous efforts on assisting variability management in systems engineering and Thales scenarios [7]. We confirm the further need of managing it at a pattern level and provide a tooling approach to model and

automate it.

The embedded systems domain is frequently subject for variability and safety assessment issues [2,9,12]. From an industrial perspective, some papers report on their experience in managing safety in a product line context [3,13]. Schulze et al. [13] demonstrated that safety-related artifacts can be treated like other artifacts and presented a comprehensive model-based tool on top of pure::variants. We are following the same direction by linking feature models with safety modeling artifacts. We provide a solution based on patterns technology for reusing modeling artefacts. We combine the patterns with variability support on top of the common variability language [8].

5. CONCLUSION

In this paper, we described the core content of a live demonstration aiming to illustrate the combination of variability and patterns in the context of a safety and security systems engineering. The demonstration will be divided into three parts and will be lively executed:

- First, we will introduce the open-source Diff/Merge pattern technology that can be used to create reusable model fragments;
- Second, we will introduce CVL [8], the KCVL[3] tooling infrastructure, and the connection to patterns;
- Finally, we will show how we can seamlessly combine these tools on a real systems engineering case study: the design of a software-defined radio family.

We illustrated the tool-supported approach on a real systems engineering and highlighted the potential benefits of combining KCVL with the patterns technology. This approach is domain-agnostic, allows industrial practitioners to define reusable model components while expressing the different valid combinations.

We are currently working on an experiment with Myriad [10], a multi-criteria decision analysis (MCDA) tool to automatically assess different architectures. Using the variability and patterns technology, we plan to explore the design space with the automatic generation of architectural variants. In another domain and context [**?**] we are also investigating the use of our tool.

Acknowledgments

The research activities were conducted in the context of Clarity and ITEA2-MERgE (Multi-Concerns Interactions System Engineering, ITEA2 11011), a European collaborative project with a focus on safety and security [4].

6. REFERENCES

[1] M. Acher. *Managing Multiple Feature Models: Foundations, Language and Applications*. PhD thesis, 2011.

[2] L. Belategi, G. Sagardui, and L. Etxeberria. Variability management in embedded product line analysis. In *Advances in System Testing and Validation Lifecycle (VALID), 2010 Second International Conference on*, pages 69–74. IEEE, 2010.

[3] R. T. V. Braga, O. Trindade, Jr., K. R. L. J. C. Branco, and J. Lee. Incorporating certification in feature modelling of an unmanned aerial vehicle product line. In *Proceedings of the 16th International Software Product Line Conference - Volume 1*, SPLC '12, pages 249–258, New York, NY, USA, 2012. ACM.

[4] T. M. consortium. Merge: Multi-concerns interactions system engineering. http://www.merge-project.eu/.

[5] D. Emery and R. Hilliard. Every architecture description needs a framework: Expressing architecture frameworks using iso/iec 42010. In *Software Architecture, 2009 & European Conference on Software Architecture. WICSA/ECSA 2009. Joint Working IEEE/IFIP Conference on*, pages 31–40. IEEE, 2009.

[6] M. Eysholdt and H. Behrens. Xtext: Implement your language faster than the quick and dirty way. In *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*, OOPSLA '10, pages 307–309, New York, NY, USA, 2010. ACM.

[7] J. B. F. Filho, O. Barais, M. Acher, J. Le Noir, A. Legay, and B. Baudry. Generating counterexamples of model-based software product lines. *International Journal on Software Tools for Technology Transfer*, pages 1–16, 2014.

[8] F. Fleurey, Ø. Haugen, B. Møller-Pedersen, A. Svendsen, and X. Zhang. Standardizing Variability - Challenges and Solutions. In *SDL Forum*, pages 233–246, 2011.

[9] R. Kolb, I. John, J. Knodel, D. Muthig, U. Haury, and G. Meier. Experiences with product line development of embedded systems at testo ag. In *Software Product Line Conference, 2006 10th International*, pages 10–pp. IEEE, 2006.

[10] C. Labreuche and F. Lehuédé. Myriad: a tool suite for mcda. *EUSFLAT'05*, pages 204–209, 2005.

[11] O. MOF. Omg meta object facility (mof) specification v1. 4, 2002.

[12] A. Polzer, S. Kowalewski, and G. Botterweck. Applying software product line techniques in model-based embedded systems engineering. In *Model-Based Methodologies for Pervasive and Embedded Software, 2009. MOMPES'09. ICSE Workshop on*, pages 2–10. IEEE, 2009.

[13] M. Schulze, J. Mauersberger, and D. Beuche. Functional safety and variability: Can it be brought together? In *Proceedings of the 17th International Software Product Line Conference*, SPLC '13, pages 236–243, New York, NY, USA, 2013. ACM.

[14] https://www.polarsys.org/projects/polarsys.capella. Capella.

[3] http://diverse-project.github.io/kcvl/